Étape 4 : Couplage à l'interface graphique

Jamila Sam & Jean-Cédric Chappelier, 2018

Version : 1.1

But

Pouvoir utiliser l'interface graphique fournie.

Description

Tester ses classes de façon systématique, au moyen de « petits » tests ponctuels ad-hoc, est une très bonne habitude (qui doit être conservée). Pouvoir tester de façon globale et observer visuellement ce qui se passe est tout aussi important. Pour cela, nous fournissons une interface graphique de simulation de notre « petit monde ».

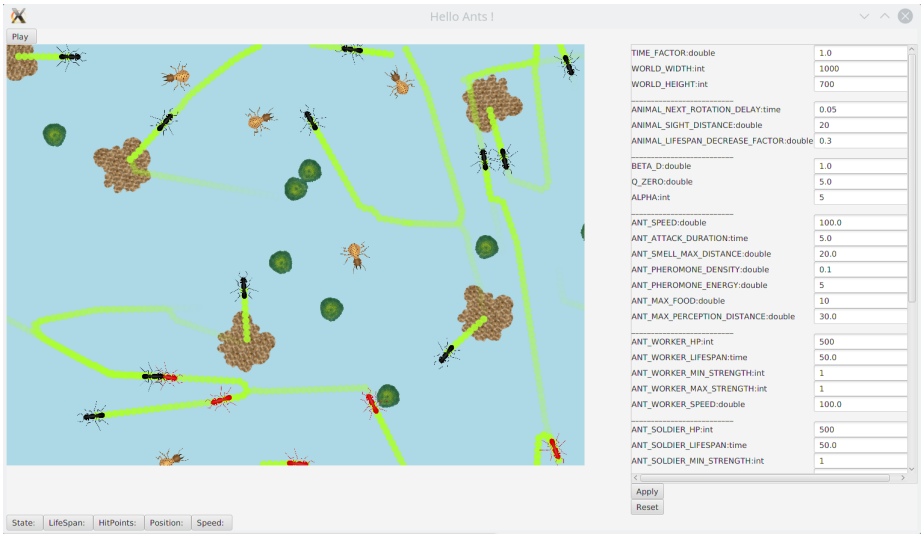

Figure 1: Interface graphique du programme de simulation

L'interface graphique, délibérément voulue comme très simple, permettra d'afficher l'ensemble des éléments de la simulation comme le montre la figure 1. Elle permettra également d'afficher à la demande des éléments d'information utiles (par exemple, à quelle quantité de nourriture correspond une `Food` donnée), ce qui peut être très utile pour vérifier le bon fonctionnement du programme.

Un peu de travail va cependant être nécessaire de votre côté pour coupler le code du projet à cette interface graphique. C'est ce que cette étape propose de réaliser.

Pour rendre le code fourni compilable, il est en fait nécessaire d'avancer un peu plus dans la mise en place de l'architecture de base de votre programme. Les éléments représentables graphiquement à savoir, les animaux, les fourmilières et la phéromone, doivent en effet être ébauchés. Seuls quelques attributs et méthodes basiques seront introduits à ce stade. La classe `Environment` doit quant à elle incorporer une méthode de rendu graphique et anticiper des fonctionnalités d'ajout d'animaux et de fourmilières.

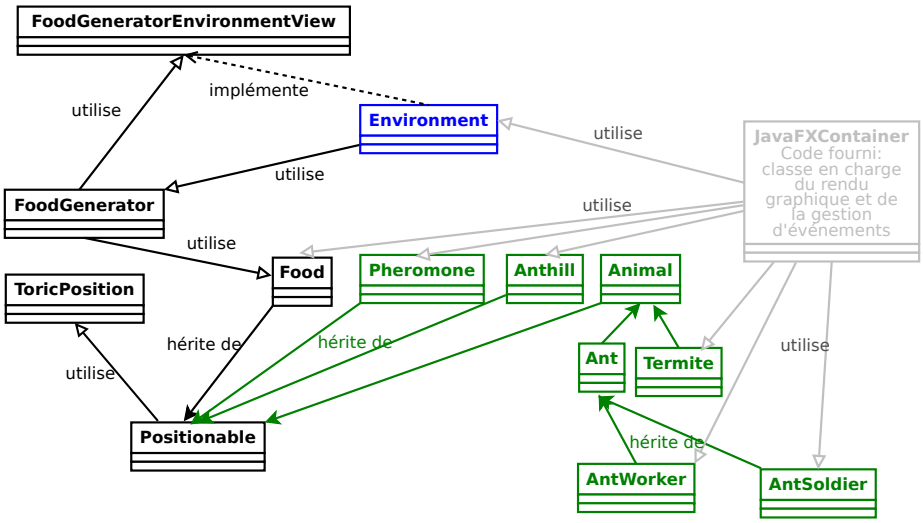

Figure 2: Architecture codée au terme de l'étape 4 (en noir les composants déjà codés, en vert les nouveaux composants, en bleu ceux retouchés)

La figure 2 donne une vue d'ensemble de l'architecture à laquelle vous allez aboutir au terme de cette étape.

Implémentation

Pour répondre aux objectifs ci-dessus, il vous est donc demandé de créer les classes suivantes :

- la classe publique et abstraite `Animal` héritant de `Positionable` ; chaque animal possède de plus un angle (de type `double`) représentant la direction dans laquelle il regarde ;

la classe `Animal` doit être dotée d'un constructeur qui initialise la position à partir d'une `ToricPosition` passée en paramètre et initialise l'angle de direction à `0.0` ;

elle doit également contenir une méthode publique finale `double getDirection()` qui retourne l'angle de direction et une méthode publique abstraite

```
void accept(AnimalVisitor visitor, RenderingMedia s)
```

simplement vide pour le moment. Cette méthode permettra le rendu graphique ddes animaux et sera expliquée plus tard dans le projet, le moment venu ;

- la classe abstraite `Ant` héritant de `Animal` : les fourmis, qu'elles soient ouvrières ou soldates, ont des propriétés communes (comme nous l'introduirons plus tard le fait d'appartenir à une fourmilière par exemple) ; il est donc naturel d'avoir une sous-classe de `Animal` spécifique aux fourmis ;
- la classe finale `Termite` héritant de `Animal`, ainsi que les classes finales `AntWorker` et `AntSoldier` héritant de `Ant` et dotées d'un constructeur prenant en paramètre une `ToricPosition` ;

les classes `Termite`, `AntWorker` et `AntSoldier` devront également implémenter la méthode

```
void accept(AnimalVisitor visitor, RenderingMedia s)
```

cette méthode fera simplement un appel à

```
visitor.visit(this, s);
```

```
**Note** : Le rendu graphique se fait en recourant à ce que l'on appelle un « _patron de co
```

- les classes finales `Anthill` et `Pheromone` héritant de `Positionable` et dotées d'un constructeur initialisant la position à l'aide d'une `ToricPosition` passée en paramètre ;

la classe `Pheromone` et la classe `Anthill` sont également dotées d'une méthode publique, *respectivement* :

```
double getQuantity()
```

et

```
double getFoodQuantity()
```

qui pour le moment retournent simplement la valeur 0.0 ;

Il vous faut également compléter la classe `Environment` avec les méthodes publiques suivantes :

```
void renderEntities(EnvironmentRenderer environmentRenderer)
void addAnthill(Anthill anthill)
void addAnimal(Animal animal)
```

La première est destinée à procéder au rendu graphique des constituants de l'environnement. Pour le moment, seules les `Food` sont modélisées de façon suffisamment complète pour être dessinées. Ceci se fait au moyen de l'instruction :

```
foods.forEach(environmentRenderer::renderFood);
```

où `foods` est la collection de `Food` de `Environment`. Cette tournure (valable depuis Java 8) signifie que l'on applique la méthode fournie `renderFood()` (dessin d'un `Food`) à chaque élément de la collection.

Le corps des méthodes `addAnthill()` et `addAnimal()` est simplement laissé vide pour le moment.

Enfin, la partie graphique a besoin de connaître les dimensions de l'environnement. Vous ajouterez donc à `Environment` les méthodes `int getWidth()` et `int getHeight()` retournant respectivement les valeurs associées au paramètres `WORLD_WIDTH` et `WORLD_HEIGHT` (valeurs obtenues comme d'habitude au moyen de `Context.getConfig()`).

Tests et soumission

Tests locaux

Vous pouvez, bien sûr, comme pour les étapes précédentes, continuer à ajouter vos propres tests dans le fichier `ch/epfl/moocprog/tests/Main.java` ; ce que nous vous encourageons à faire. Mais le but premier de cette étape ci est de pouvoir utiliser l'interface graphique fournie. C'est donc ce point là que nous allons expliquer maintenant.

La version graphique du programme principal de simulation se trouve dans :

`ch/epfl/moocprog/app/Main.java`

Pour le lancer, il suffit de faire un clic droit dans Eclipse puis `Run as > Java application`.

Vous devriez alors voir apparaître la représentation graphique de l'environnement (rectangle bleu). La simulation est par défaut en mode «pause». Pour la lancer,

Figure 3: Information de «debugging»

appuyez sur le bouton `Play` en haut à gauche. Vous devriez alors voir spontanément apparaître dans l'environnement des tas de nourriture aléatoirement distribuées (petits tas vert donc chacun représente une instance de `Food`).

Certaines informations utiles au debugging peuvent être obtenues au moyen des boutons se trouvant juste au dessous du rectangle bleu.

Par exemple, pour examiner et vérifier les quantités de nourriture présentes dans chaque tas, vous pouvez cliquer sur le bouton `Quantity`. De même, la position des tas peut s'afficher au moyen du bouton `Position`. Un exemple d'affichage ainsi obtenu est donné par la figure 3. Cliquer à nouveau sur le bouton `Quantity` ou `Position` causera l'effacement des informations de debugging textuelles relatives (jouez avec ces boutons pour constater leur effet, les autres boutons sont pour le moment sans impact).

Vous pouvez à tout moment recommencer la simulation depuis zéro (réinitialisation de l'environnement) en utilisant le bouton `Restart` ou en stopper momentanément le déroulement au moyen du bouton `Pause`. Ceci permet d'observer tranquillement les informations de debugging liés aux objets affichés avant de poursuivre la simulation.

Les paramètres éditables sur la droite sont à ce stade sans effet à part `TIME_FACTOR` ou `WORLD_WIDTH` et `WORLD_HEIGHT`.

Changez la valeur de `TIME_FACTOR` à 5 par exemple puis appuyez sur `Apply`. Vous devriez voir la simulation se dérouler beaucoup plus vite. Cliquez sur `Reset` pour retrouver la valeur d'origne de `TIME_FACTOR`.

Rappelons que le rôle des paramètres de simulation est expliqué dans le document

« *Complément : paramètres de simulation* »de la semaine 1. Les modifications faites aux paramètres sont directement écrites dans le fichier `res/app.cfg`.

Si vous voulez retrouver les valeurs de départ lors du prochain lancement du programme, il ne faut pas oublier de recourir au bouton `Reset` avant de quitter le programme ou préalablement à un `Restart`.

Notez enfin que la modification des valeurs `WORLD_WIDTH` et `WORLD_HEIGHT` (taille graphique de l'environnement) ne prend effet qu'une fois que l'on a quitté puis relancé le programme. Il en sera ainsi pour tous les paramètres qui sont directement utilisés par les constructeurs des objets de la simulation.

Un peu de décryptage

Voici, à toutes fins utiles, une petite explication de ce qui se passe lorsque vous lancez le programme graphique de simulation :

- Le programme de simulation correspond à un objet `Application` associé à un gestionnaire de paramètres (`ConfigManager`) lui même lié à un contexte d'utilisation (un objet `Context` qui indique quel fichier de paramétrage est employé par le gestionnaire de paramètres).
- Au programme de simulation est associé un composant JavaFX (`gfx/JavaFXContainer`) qui permet de produire des représentations graphiques et de répondre à des événements extérieurs (clics de boutons par l'utilisateur).
- L'élément fondamental de `JavaFXContainer` est la boucle de simulation, la méthode `handle`. Cette dernière appelle en boucle les méthodes `update` et `renderEntities` de votre classe `Environment`. C'est par ce biais que ce fait le lien entre le matériel fourni et ce que vous créez de votre côté.

Soumission

Pour soumettre votre devoir au correcteur automatique, il faut créer un fichier ZIP contenant **TOUS VOS** fichiers sources, depuis la racine `ch`, ceux de cette étape mais aussi ceux de l'étape précédente. Concrètement, pour ce devoir ci, votre fichier ZIP doit donc contenir exactement les fichiers suivants :

```
ch/epfl/moocprog/Animal.java
ch/epfl/moocprog/Anthill.java
ch/epfl/moocprog/Ant.java
ch/epfl/moocprog/AntSoldier.java
ch/epfl/moocprog/AntWorker.java
ch/epfl/moocprog/Environment.java
ch/epfl/moocprog/FoodGeneratorEnvironmentView.java
ch/epfl/moocprog/FoodGenerator.java
```

`ch/epfl/moocprog/Food.java`
`ch/epfl/moocprog/Pheromone.java`
`ch/epfl/moocprog/Positionable.java`
`ch/epfl/moocprog/Termite.java`
`ch/epfl/moocprog/ToricPosition.java`

Note : si c'est plus pratique pour vous, vous *pouvez* aussi faire un zip qui contient tout le sous-répertoire `ch/epfl/moocprog`, y compris, donc, les fichiers que nous vous avons fournis. Ces fichiers seront simplement ignorés par le correcteur automatique (et remplacés par les nôtres). L'énoncé « *Procédure de soumission* » de la semaine 1 vous indique comment procéder. **Avant de passer à l'étape suivante, n'oubliez pas de faire une sauvegarde de l'étape en cours**, comme indiqué dans le même document.